On Implementation and Evaluation of Inverse Iteration Algorithm with compact WY Orthogonalization

Hiroyuki Ishigami, Kinji Kimura, Yoshimasa Nakamura

Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Sakyo-ku, Kyoto 606 8501, Japan

Abstract

A new inverse iteration algorithm that can be used to compute all the eigenvectors of a real symmetric tri-diagonal matrix on parallel computers is developed. The modified Gram-Schmidt orthogonalization is used in the classical inverse iteration. This algorithm is sequential and causes a bottleneck in parallel computing. In this paper, the use of the compact WY representation is proposed in the orthogonalization process of the inverse iteration with the Householder transformation. This change results in drastically reduced synchronization cost in parallel computing. The new algorithm is evaluated on both an 8-core and a 32-core parallel computer, and it is shown that the new algorithm is greatly faster than the classical inverse iteration algorithm in computing all the eigenvectors of matrices with several thousand dimensions.

Keywords: inverse iteration, orthogonalization, compact WY representation, eigenvalue problem, parallelization, Householder transformation

1. Introduction

The eigenvalue decomposition of a symmetric matrix, i.e., a decomposition into a product of matrices consisting of eigenvectors and eigenvalues, is one of the most important operations in linear algebra. It is used in vibrational analysis, image processing, data searches, etc.

Let us note that the eigenvalue decomposition of real symmetric matrices is reduced to that of real symmetric tri-diagonal matrices. Owing to recent improvements in the performance of computers equipped with multicore processors, we have had more opportunities to perform computation on parallel computers. As a result, there has been an increase in demand for an eigenvalue decomposition algorithm that can be effectively parallelized.

The inverse iteration algorithm is an algorithm for computing eigenvectors independently associated with mutually distinct eigenvalues. However, when we use this algorithm, we must reorthogonalize the eigenvectors if some eigenvalues are very close

Email addresses: hishigami@amp.i.kyoto-u.ac.jp (Hiroyuki Ishigami), kkimur@amp.i.kyoto-u.ac.jp (Kinji Kimura), ynaka@i.kyoto-u.ac.jp (Yoshimasa Nakamura)

to each other. Adding this reorthogonalization process increases the computational cost. For this reorthogonalization, we have generally used the MGS (modified Gram-Schmidt) algorithm. However, this algorithm is sequential and inefficient for parallel computing. As a result, we are unable to maximize the performance of parallel computers. Hereinafter, we will refer to the inverse iteration algorithm with MGS as the classical inverse iteration.

We can also orthogonalize vectors by using the Householder transformation [10] and we call this precess the Householder orthogonalization algorithm. While the MGS algorithm is unstable in the sense that the orthogonality of the resulting vectors crucially depends on the condition number of the matrix [11], the Householder algorithm is stable because its orthogonality does not depend on the condition number. The Householder algorithm is also sequential and ineffective for parallel computing, and its computational cost is higher than that of MGS.

In 1989, the Householder orthogonalization in terms of the compact WY representation was proposed by R. Schreiber *et al* [9]. By adopting this orthogonalization, stability and effective parallelization can be achieved. Hereafter, we refer to this algorithm as the compact WY orthogonalization algorithm. Yamamoto *et al.* [11] reformulated this algorithm for an incremental orthogonalization. Moreover, They showed that this algorithm achieves theoretically high accurate orthogonality and high scalability in parallel computing [11]. Here, the incremental orthogonalization is implemented on many numerical computation library. LAPACK(Linear Algebra PACKage) [7] is one of the most popular libraries and all the code of LAPACK is implemented by using BLAS (Basic Linear Algebra Subroutines) operations. The compact WY orthogonalization algorithm can be implemented by using BLAS.

In [6], authors have implemented the compact WY orthogonalization to the reorthogonalization process of inverse iteration for computing eigenvectors of a tri-diagonal matrix. It is shown [6] that, in parallel computing, the new inverse iteration algorithm is faster than the classical one.

In this paper, we present two implementations: One is a new implementation of the compact WY orthogonalization algorithm based on BLAS. We focus on a mathematical structure of this algorithm and reformulate this algorithm. Therefore, using this new implementation, the computational cost of the compact WY orthogonalization can be reduced. The other is an implementation of the compact WY orthogonalization to the inverse iteration algorithm for a real symmetric tri-diagonal matrix. Thereafter, we perform the numerical experiments by computing all the eigenvectors using the second implementation and evaluate its performance.

2. Classical inverse iteration and its defect

2.1. Classical inverse iteration

We consider the problem of computing eigenvectors of a real symmetric tri-diagonal matrix $T \in \mathbb{R}^{n\times n}$. Let $\lambda_j \in \mathbb{R}$ be eigenvalues of T such that $\lambda_1 < \lambda_2 < \cdots < \lambda_n$. Let $\boldsymbol{v}_j \in \mathbb{R}^n$ be the eigenvector associated with λ_j. When $\tilde{\lambda}_j$, an approximate value of λ_j, and a starting vector $\boldsymbol{v}_j^{(0)}$ are given, we can compute an eigenvectors of T. To this end,

Alg. 1 Classical inverse iteration

```
1: for j = 1 to n do
2:     Generate v_j^(0) from random numbers.
3:     k = 0.
4:     repeat
5:         k ← k + 1.
6:         Normalize v_j^(k−1).
7:         Solve (T − λ̃_j I) v_j^(k) = v_j^(k−1) (Eq.(1)).
8:         if then|λ̃_j − λ̃_{j−1}| ≤ 10^{−3}‖T‖,
9:             for i = j_1 to j − 1 do
10:                v_j^(k) ← v_j^(k) − ⟨v_j^(k), v_i⟩ v_i
11:            end for
12:        else
13:            j_1 = j.
14:        end if
15:    until some condition is met.
16:    Normalize v_j^(k) to v_j.
17: end for
```

we solve the following equation iteratively:

$$\left(T - \tilde{\lambda}_j I\right) \boldsymbol{v}_j^{(k)} = \boldsymbol{v}_j^{(k-1)}. \tag{1}$$

Here I is the n-dimensional identity matrix. If the eigenvalues of T are mutually well-separated, $\boldsymbol{v}_j^{(k)}$, the solution of Eq.(1), generically converges to the eigenvector associated with λ_j as k goes to ∞. The above iteration method is the inverse iteration. The computational cost of this method is of $O(mn)$ when we compute m eigenvectors. In the implementation, we have to normalize the vectors $\boldsymbol{v}_j^{(k)}$ to avoid overflow.

When some of the eigenvalues are close to each other or there are clusters of eigenvalues of T, we have to reorthogonalize all the eigenvectors associated with such eigenvalues because they need to be orthogonal to each other. In the classical inverse iteration, we apply the MGS to this process and the computational cost of it is of $O(m^2 n)$. Therefore, when we compute eigenvectors of the matrix that has many clustered eigenvalues, the total computational cost increases significantly. In addition, the classical inverse iteration is implemented the Peters-Wilkinson method [8]. In this method, when the distance between the close eigenvalues is less than $10^{-3}\|T\|$, we regard them as members of the same cluster of eigenvalues, and we orthogonalize all of the eigenvectors associated with these eigenvalues. The classical inverse iteration algorithm is shown by Alg.1, and j_1 denotes the index of the minimum eigenvalue of some cluster. This algorithm is implemented as DSTEIN in LAPACK [7].

2.2. The defect of the classical inverse iteration

The inverse iteration is a prominent method for computing eigenvectors, because we can compute eigenvectors independently. When there are many clusters in the dis-

Alg. 2 Householder orthogonalization

1: **for** $j = 1$ to m **do**
2: $\quad \boldsymbol{u}_j \leftarrow \left(I - t_1 \boldsymbol{y}_1 \boldsymbol{y}_1^\top\right) \boldsymbol{v}_j$
3: $\quad$ **for** $i = 2$ to $j-1$ **do**
4: $\quad\quad \boldsymbol{u}_j \leftarrow \left(I - t_i \boldsymbol{y}_i \boldsymbol{y}_i^\top\right) \boldsymbol{u}_j$
5: $\quad$ **end for**
6: $\quad$ Compute $\boldsymbol{y}_j$ and t_j by using $\boldsymbol{u}_j$
7: $\quad \boldsymbol{q}_j \leftarrow \left(I - t_j \boldsymbol{y}_j \boldsymbol{y}_j^\top\right) \boldsymbol{e}_j$
8: $\quad$ **for** $i = j-1$ to 1 **do**
9: $\quad\quad \boldsymbol{q}_j \leftarrow \left(I - t_i \boldsymbol{y}_i \boldsymbol{y}_i^\top\right) \boldsymbol{q}_j$
10: $\quad$ **end for**
11: **end for**

tribution of eigenvalues, the inverse iteration can be parallelized by assigning each cluster to each core.

Let us consider the Peters-Wilkinson method in the classical inverse iteration. When the dimension of T is greater than 1000, most of the eigenvalues are regarded as being in the same cluster [3]. In this case, we have to parallelize the inverse iteration with respect to not the cluster but the loop described from lines 2 to 16 in Alg.1. This loop includes the iteration based on Eq.(1) and the orthogonalization of the eigenvectors. This orthogonalization process becomes a bottleneck of the classical inverse iteration with respect to the computational cost. The MGS algorithm is mainly based on a BLAS level-1 operation and it is a sequential algorithm. Because of this, when we compute all the eigenvectors on parallel computers, the number of synchronizations is of $O(m^2)$. Therefore, the MGS algorithm is ineffective in parallel computing.

In conclusion, the classical inverse iteration is an ineffective algorithm for parallel computing because the MGS algorithm is used in its orthogonalization process.

3. Other orthogonalization algorithms

In this section, we introduce alternative orthogonalization algorithms instead of the MGS algorithm. Now, we discuss the incremental orthogonalization of $\boldsymbol{v}_j \in \mathbb{R}^n$ to $\boldsymbol{q}_j \in \mathbb{R}^n$ ($j = 1, \ldots, m$, $m \leq n$). The incremental orthogonalization arises in the reorthogonalization process on the inverse iteration and it is defined as follows: $\boldsymbol{v}_j$ ($2 \leq j \leq m$) is not given in advance but is computed from $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$.

In the following, Let us define a vector $\boldsymbol{0}_i$ as the i-dimensional zero vector and matrices $V, Q \in \mathbb{R}^{n \times m}$ as $V = [\boldsymbol{v}_1 \ \cdots \ \boldsymbol{v}_m]$, $Q = [\boldsymbol{q}_1 \ \cdots \ \boldsymbol{q}_m]$.

3.1. Householder orthogonalization

The Householder orthogonalization, based on the Householder matrices, is one of the alternative orthogonalization methods. When vectors $\boldsymbol{u}_j, \boldsymbol{w}_j \in \mathbb{R}^n$ ($j = 1, \ldots, m$) satisfy $\|\boldsymbol{u}_j\|_2 = \|\boldsymbol{w}_j\|_2$, there exists the orthogonal matrices H_j called the Householder matrices satisfying $H_j H_j^\top = H_j^\top H_j = I$, $H_j \boldsymbol{u}_j = \boldsymbol{w}_j$ defined by $H_j = I - t_j \boldsymbol{y}_j \boldsymbol{y}_j^\top$, $\boldsymbol{y}_j = \boldsymbol{u}_j - \boldsymbol{w}_j$, $t_j = 2/\|\boldsymbol{y}_j\|_2^2$. The transformation from $\boldsymbol{u}_j$ to $\boldsymbol{v}_j$ by H_j is called the Householder

transformation. By using the Householder transformations. This orthogonalization algorithm is shown in Alg.2. The vector $\boldsymbol{y}_j$ is the vector in which the elements from 1 to $(j-1)$ are the same as the elements of $\boldsymbol{u}_j$ and the elements from $(j+1)$ to n are zero. The vectors $\boldsymbol{u}_j$ and $\boldsymbol{w}_j$ are defined as follows:

$$\begin{aligned}\boldsymbol{u}_j &= \begin{bmatrix} u_{1,j} & \cdots & u_{j-1,j} & u_{j,j} & u_{j+1,j} & \cdots & u_{n,j} \end{bmatrix}^\top \\ &= H_{j-1}H_{j-2}\cdots H_2 H_1 \boldsymbol{v}_j, \\ \boldsymbol{w}_j &= \begin{bmatrix} u_{1,j} & \cdots & u_{j-1,j} & c_j & \boldsymbol{0}_{n-j}^\top \end{bmatrix}^\top,\end{aligned}$$

where $u_{i,j}$ $(i = 1, \ldots, n)$ is the i-th element of $\boldsymbol{u}_j$ and

$$c_j = -\operatorname{sgn}(u_{j,j}) \sqrt{\sum_{i=j}^{n} u_{i,j}^2}.$$

Here, $\boldsymbol{y}_j$ and t_j are computed as follows:

$$\boldsymbol{y}_j = \boldsymbol{u}_j - \boldsymbol{w}_j = \begin{bmatrix} \boldsymbol{0}_{j-1}^\top & u_{j,j} - c_j & u_{j+1,j} & \cdots & u_{n,j} \end{bmatrix}^\top, t_j = \frac{2}{\|\boldsymbol{y}_j\|_2^2}. \tag{2}$$

The vector $\boldsymbol{e}_j$ in Alg.2 is the j-th vector of an n-dimensional identity matrix.

The orthogonality of the vectors $\boldsymbol{q}_j$ generated by the Householder orthogonalization does not depend on the condition number of V. Therefore, the Householder orthogonalization is more stable than MGS. On the other hand, being similar to MGS, it is a sequential algorithm, that is mainly based on a BLAS level-1 operation. Its computational cost is about twice higher than that of MGS. Thus the Householder orthogonalization is an ineffective algorithm for parallel computing.

3.2. Compact WY orthogonalization

In 1989, the Householder orthogonalization in terms of the compact WY representation was proposed by Schreiber and van Loan [9]. Yamamoto and Hirota [11] reformulated this algorithm for the incremental orthogonalization. This study suggests that the Householder orthogonalization becomes capable of computation with a BLAS level-2 operation in terms of the compact WY representation. They also showed that this algorithm achieved theoretically high orthogonality and high scalability in parallel computing [11].

Now, we consider the Householder orthogonalization in Alg.2 and we introduce the compact WY representation. First, we define $Y_1 = [\boldsymbol{y}_1] \in \mathbb{R}^{n\times 1}$ and $T_1 = [t_1] \in \mathbb{R}^{1\times 1}$. Let us define matrices $Y_j \in \mathbb{R}^{n\times j}$ and upper triangular matrices $T_j \in \mathbb{R}^{j\times j}$ recursively as follows:

$$Y_j = \begin{bmatrix} Y_{j-1} & \boldsymbol{y}_j \end{bmatrix}, \quad T_j = \begin{bmatrix} T_{j-1} & -t_j T_{j-1} Y_{j-1}^\top \boldsymbol{y}_j \\ \boldsymbol{0}_{j-1}^\top & t_j \end{bmatrix}. \tag{3}$$

In this case, the following equation holds

$$H_1 H_2 \cdots H_j = I - Y_j T_j Y_j^\top. \tag{4}$$

As shown in Eq.(4), we can rewrite the product of the Householder matrices $H_1 H_2 \cdots H_j$ in a simple block matrix form. Here $I - Y_j T_j Y_j^\top$ is called the compact WY representation of the product $H_1 H_2 \cdots H_j$ of the Householder matrices. Alg.3 shows the compact WY orthogonalization algorithm.

Alg. 3 compact WY orthogonalization algorithm

1: Compute $\boldsymbol{y}_1$ and t_1 by using $\boldsymbol{u}_1 = \boldsymbol{v}_1$
2: $Y_1 = [\boldsymbol{y}_1]$, $T_1 = [t_1]$
3: $\boldsymbol{q}_1 \leftarrow \left(I - Y_1 T_1 Y_1^\top\right) \boldsymbol{e}_j$
4: **for** $j = 2$ to m **do**
5: $\quad \boldsymbol{u}_j \leftarrow \left(I - Y_{j-1} T_{j-1}^\top Y_{j-1}^\top\right) \boldsymbol{v}_j$
6: $\quad$ Compute $\boldsymbol{y}_j$ and t_j by using $\boldsymbol{u}_j$
7: $\quad Y_j = \begin{bmatrix} Y_{j-1} & \boldsymbol{y}_j \end{bmatrix}$, $T_j = \begin{bmatrix} T_{j-1} & -t_j T_{j-1} Y_{j-1}^\top \boldsymbol{y}_j \\ \boldsymbol{0} & t_j \end{bmatrix}$.
8: $\quad \boldsymbol{q}_j \leftarrow \left(I - Y_j T_j Y_j^\top\right) \boldsymbol{e}_j$
9: **end for**

3.3. Implementation of compact WY orthogonalization

In this subsection, we discuss the implementation of the compact WY orthogonalization algorithm using BLAS operations. In addition, we discuss a mathematical structure of this algorithm and present a new implementation of the compact WY orthogonalization for reducing the computational cost and the usage of memory.

3.3.1. Ordinary implementation of compact WY orthogonalization using BLAS

Now we discuss the implementation of the compact WY orthogonalization based on line 5 to 8 in Alg.3 using BLAS operations.

For the adaptation of BLAS operations, we have to reformulate the formula of line 5 as follows:

$$\begin{aligned} \boldsymbol{u}_j &= \left(I - Y_{j-1} T_{j-1}^\top Y_{j-1}^\top\right) \boldsymbol{v}_j \\ &= \boldsymbol{v}_j - Y_{j-1} T_{j-1}^\top Y_{j-1}^\top \boldsymbol{v}_j \end{aligned}$$

Now we can implement this formula by using BLAS as follows:

$$\begin{cases} \boldsymbol{u}_j \leftarrow \boldsymbol{v}_j & \text{(DCOPY)} \\ \boldsymbol{v}'_{j-1} \leftarrow Y_{j-1}^\top \boldsymbol{u}_j + 0 \cdot \boldsymbol{v}'_{j-1} & \text{(DGEMV)} \\ \boldsymbol{v}'_{j-1} \leftarrow T_{j-1}^\top \boldsymbol{v}'_{j-1} & \text{(DTRMV)} \\ \boldsymbol{u}_j \leftarrow (-1) \cdot Y_{j-1} \boldsymbol{v}'_{j-1} + \boldsymbol{u}_j & \text{(DGEMV)} \end{cases},$$

where $\boldsymbol{v}'_{j-1} \in \mathbb{R}^{j-1}$. We set the initial address of $\boldsymbol{v}'_{j-1}$ assigned on CPU memory to correspond to that of $\boldsymbol{v}_j$. DCOPY denotes the copying operation of a vector $\boldsymbol{x}$ to a vector $\boldsymbol{y}$: $\boldsymbol{y} \leftarrow \boldsymbol{x}$. DGEMV means the matrix-vector operation: $\boldsymbol{y} \leftarrow \alpha A \boldsymbol{x} + \beta \boldsymbol{y}$, where A is a general rectangular matrix. DTRMV denotes the matrix-vector product: $\boldsymbol{x} \leftarrow T \boldsymbol{x}$, where T is a triangular matrix.

Next, on line 6, we compute $\boldsymbol{y}_j$ and t_j based on Eq.(2). These computations is mainly performed by using BLAS level-1 operations and its computational cost is rel-

atively lower. we implement the computation of $\boldsymbol{y}_j$ and t_j as follows:

$$\begin{cases} y_{i,j} \leftarrow 0,\ (i = 1, \ldots,\ j-1) & \\ y_{i,j} \leftarrow u_{i,j},\ (i = j, \ldots, n) & \text{(DCOPY)} \\ y_{j,j} \leftarrow u_{j,j} - c_j,\ \ c_j = -\operatorname{sgn}(u_{j,j}) \sqrt{\sum_{i=j}^{n} u_{i,j}^2} & \text{(DNRM2)} \\ t_j \leftarrow 2/\|\boldsymbol{y}_j\|_2^2 & \text{(DNRM2)} \end{cases},$$

where $y_{i,j}$ $(i = 1, \ldots, n)$ is the i-th column element of $\boldsymbol{y}_j$. DNRM2 denotes the computation of the 2-norm of a vector.

On line 7, updating Y_j and t_j can be done easily. Now, let $\hat{\boldsymbol{t}}_j \in \mathbb{R}^{j-1}$ be $\hat{\boldsymbol{t}}_j = -t_j T_{j-1} Y_{j-1}^\top \boldsymbol{y}_j$. Note that $\hat{\boldsymbol{t}}_j$ is implemented by using BLAS as follows:

$$\begin{cases} \hat{\boldsymbol{t}}_j \leftarrow (-t_j) Y_{j-1}^\top \boldsymbol{y}_j + 0 \cdot \hat{\boldsymbol{t}}_j & \text{(DGEMV)} \\ \hat{\boldsymbol{t}}_j \leftarrow T_{j-1} \hat{\boldsymbol{t}}_j & \text{(DTRMV)} \end{cases}.$$

At last, on line 8, we can reformulate as follows:

$$\begin{aligned} \boldsymbol{q}_j &= \left(I - Y_j T_j Y_j^\top\right) \boldsymbol{e}_j \\ &= \boldsymbol{e}_j - Y_j T_j Y_j^\top \boldsymbol{e}_j. \end{aligned}$$

Here, the matrix-vector product $Y_j^\top \boldsymbol{e}_j$ can be simplified as follows:

$$Y_j^\top \boldsymbol{e}_j = \begin{bmatrix} y_{j,1} \\ \vdots \\ y_{j,j} \end{bmatrix}$$

. This computation can be performed only by copying the j-th column of Y_j to some vector. Therefore we can implement the formula of line 8 using BLAS as follows:

$$\begin{cases} \boldsymbol{q}_j \leftarrow \boldsymbol{e}_j & \text{(DCOPY)} \\ \boldsymbol{v}'_j \leftarrow \begin{bmatrix} y_{j,1} & \cdots & y_{j,j} \end{bmatrix} & \text{(DCOPY)} \\ \boldsymbol{v}'_j \leftarrow T_j^\top \boldsymbol{v}'_j & \text{(DTRMV)} \\ \boldsymbol{q}_j \leftarrow (-1) \cdot Y_j \boldsymbol{v}'_j + \boldsymbol{q}_j & \text{(DGEMV)} \end{cases},$$

where $\boldsymbol{v}'_j \in \mathbb{R}^j$, $\boldsymbol{q}_j \in \mathbb{R}^n$. We set the initial address of $\boldsymbol{v}'_j$, $\boldsymbol{q}_j$ assigned on CPU memory to correspond to that of $\boldsymbol{u}_j$, $\boldsymbol{v}_j$, respectively.

The computational cost of the above compact WY orthogonalization algorithm is almost $4m^2n + m^3$. In the worst case, i.e., $m = n$, the computational cost is $5n^3$.

In addition, for this implementation, we have to use almost $mn + m^2$ CPU memory because Y_m use mn and T_m use m^2 domain.

3.3.2. New implementation of compact WY orthogonalization using BLAS

In the above section, we discuss the ordinary implementation of the compact WY orthogonalization algorithm. Now we focus on the mathematical structure of this algorithm and present the new implementation of the compact WY orthogonalization which has the less computational cost than the ordinary one has.

Before the formula of line 5 in Alg.3, let us consider the formula of line 6. From Eq.(2), we can strictly compute t_j as follows: Since

$$c_j = -\operatorname{sgn}(u_{j,j}) \sqrt{\sum_{i=j}^{n} u_{i,j}^2},$$

we have

$$\begin{aligned}
\|\boldsymbol{y}_j\|_2^2 &= (u_{j,j} - c_j)^2 + \sum_{i=j+1}^{n} u_{i,j}^2 \\
&= \sum_{i=j}^{n} u_{i,j}^2 - 2u_{j,j}c_j + c_j^2 \\
&= 2(c_j^2 - u_{j,j}c_j).
\end{aligned}$$

Hence, we have

$$t_j = \frac{2}{\|\boldsymbol{y}_j\|_2^2} = \frac{1}{c_j^2 - u_{j,j}c_j}.$$

From this fact and the definition of $\boldsymbol{y}_j$ and c_j, we need not compute the elements from 1 to $(j-1)$ of $\boldsymbol{u}_j$ in actual. Therefore we compute only the elements from j to n of $\boldsymbol{u}_j$ so that the formula of line 5 is reduced as follows:

$$\hat{\boldsymbol{u}}_j = \hat{\boldsymbol{u}}_j - \hat{Y}_{j-1} T_{j-1}^{\top} Y_{j-1}^{\top} \boldsymbol{v}_j,$$

where $\hat{\boldsymbol{u}}_j \in \mathbb{R}^{n-(j-1)}$ is $\hat{\boldsymbol{u}}_j = \begin{bmatrix} u_{j,j} & \cdots & u_{n,j} \end{bmatrix}^{\top}$.

Here, we focus on the structure of $\boldsymbol{y}_j$. From Eq.(2), $\boldsymbol{y}_j$ ($j = 2, \ldots, m$) can be represented as the block vector of the form:

$$\boldsymbol{y}_j = \begin{bmatrix} \boldsymbol{0}_{j-1} \\ \hat{\boldsymbol{y}}_j \end{bmatrix},$$

where $\hat{\boldsymbol{y}}_j \in \mathbb{R}^{n-(j-1)}$ is the vector of nonzero elements of $\boldsymbol{y}_j$. From this fact, Y_j can be represented as the following block matrix:

$$Y_j = \begin{bmatrix} L_j \\ \hat{Y}_j \end{bmatrix},$$

where $L_j \in \mathbb{R}^{j \times j}$ is a lower triangular matrix and $\hat{Y}_j \in \mathbb{R}^{(n-j) \times j}$ is generally a dense rectangular matrix. In addition, let us consider $\boldsymbol{v}_j$ as the block vector of the form:

$$\boldsymbol{v}_j = \begin{bmatrix} \check{\boldsymbol{v}}_j \\ \hat{\boldsymbol{v}}_j \end{bmatrix},$$

where $\check{\boldsymbol{v}}_j \in \mathbb{R}^{j-1}$, $\hat{\boldsymbol{v}}_j \in \mathbb{R}^{n-(j-1)}$.

By using these block form of $\boldsymbol{v}_j$ and Y_j, we can reduce the computational cost of the matrix-vector product $Y_{j-1}^\top \boldsymbol{v}_j$ through

$$Y_{j-1}^\top \boldsymbol{v}_j = \begin{bmatrix} L_{j-1} \\ \hat{Y}_{j-1} \end{bmatrix}^\top \begin{bmatrix} \check{\boldsymbol{v}}_j \\ \hat{\boldsymbol{v}}_j \end{bmatrix} = L_{j-1}^\top \check{\boldsymbol{v}}_j + \hat{Y}_{j-1}^\top \hat{\boldsymbol{v}}_j.$$

Therefore, the formula of $\hat{\boldsymbol{u}}_j$ can be simplified as follows:

$$\hat{\boldsymbol{u}}_j = \hat{\boldsymbol{u}}_j - \hat{Y}_{j-1} T_{j-1}^\top \left(L_{j-1}^\top \check{\boldsymbol{v}}_j + \hat{Y}_{j-1}^\top \hat{\boldsymbol{v}}_j \right).$$

This formula can be implemented by using BLAS as follows:

$$\begin{cases} \hat{\boldsymbol{u}}_j \leftarrow \hat{\boldsymbol{v}}_j & \text{(DCOPY)} \\ \check{\boldsymbol{v}}_j \leftarrow L_{j-1}^\top \check{\boldsymbol{v}}_j & \text{(DTRMV)} \\ \check{\boldsymbol{v}}_j \leftarrow \hat{Y}_{j-1}^\top \hat{\boldsymbol{v}}_j + \check{\boldsymbol{v}}_j & \text{(DGEMV)} \\ \check{\boldsymbol{v}}_j \leftarrow T_{j-1}^\top \check{\boldsymbol{v}}_j & \text{(DTRMV)} \\ \hat{\boldsymbol{u}}_j \leftarrow (-1) \cdot \hat{Y}_{j-1} \check{\boldsymbol{v}}_j + \hat{\boldsymbol{u}}_j & \text{(DGEMV)} \end{cases}.$$

From the above discussion, the computation on line 6 is implemented by using BLAS as follows:

$$\begin{cases} y_{i,j} \leftarrow u_{i,j},\ (i = j, \ldots, n) & \text{(DCOPY)} \\ y_{j,j} \leftarrow u_{j,j} - c_j,\ \ c_j = -\operatorname{sgn}(u_{j,j}) \sqrt{\sum_{i=j}^n u_{i,j}^2} & \text{(DNRM2)} \\ t_j \leftarrow 1 / \left(c_j^2 - u_{j,j} c_j \right) & \end{cases}.$$

On line 7, we can also reduce the computational cost of $\hat{\boldsymbol{t}}_j$ through

$$\begin{aligned} \hat{\boldsymbol{t}}_j &= -t_j T_{j-1} Y_{j-1}^\top \boldsymbol{y}_j \\ &= -t_j T_{j-1} \begin{bmatrix} L_{j-1} \\ \hat{Y}_{j-1} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{0}_{j-1} \\ \hat{\boldsymbol{y}}_j \end{bmatrix} \\ &= -t_j T_{j-1} \left(L_{j-1}^\top \boldsymbol{0}_{j-1} + \hat{Y}_{j-1}^\top \hat{\boldsymbol{y}}_j \right) \\ &= -t_j T_{j-1} \hat{Y}_{j-1}^\top \hat{\boldsymbol{y}}_j. \end{aligned}$$

This formula can be implemented by using BLAS as follows:

$$\begin{cases} \hat{\boldsymbol{t}}_j \leftarrow (-t_j) \hat{Y}_{j-1}^\top \hat{\boldsymbol{y}}_j + 0 \cdot \hat{\boldsymbol{t}}_j & \text{(DGEMV)} \\ \hat{\boldsymbol{t}}_j \leftarrow T_{j-1} \hat{\boldsymbol{t}}_j & \text{(DTRMV)} \end{cases}.$$

At last, on line 8, even if the sign of the orthogonal vector $\boldsymbol{q}_j$ is reversed, the orthogonality along with other vectors is not changed. Therefore, we can reformulate $\boldsymbol{q}_j$ as $\boldsymbol{q}_j = \left(Y_j T_j Y_j^\top - I \right) \boldsymbol{e}_j$. In addition, let us consider $\boldsymbol{q}_j$ as the following block vector:

$$\boldsymbol{q}_j = \begin{bmatrix} \check{\boldsymbol{q}}_j \\ \hat{\boldsymbol{q}}_j \end{bmatrix},$$

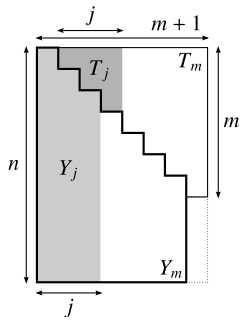

Figure 1: Assignment model for Y_j and T_j

where $\check{\boldsymbol{q}}_j \in \mathbb{R}^j$, $\hat{\boldsymbol{q}}_j \in \mathbb{R}^{n-j}$. These are reformulated as follows:

$$\begin{bmatrix} \check{\boldsymbol{q}}_j \\ \hat{\boldsymbol{q}}_j \end{bmatrix} = \begin{bmatrix} L_j T_j Y_j^\top \boldsymbol{e}_j \\ \hat{Y}_j T_j Y_j^\top \boldsymbol{e}_j \end{bmatrix} - \begin{bmatrix} \check{\boldsymbol{e}}_j \\ \boldsymbol{0}_{n-j} \end{bmatrix},$$

where $\check{\boldsymbol{e}}_j$ is the j-th vector of the j-dimensional identity matrix. Therefore this formula can be implemented by using BLAS as follows:

$$\begin{cases} \boldsymbol{x}_j \leftarrow \begin{bmatrix} y_{j,1} & \cdots & y_{j,j} \end{bmatrix} & \text{(DCOPY)} \\ \boldsymbol{x}_j \leftarrow T_j^\top \boldsymbol{x}_j & \text{(DTRMV)} \\ \check{\boldsymbol{q}}_j \leftarrow \boldsymbol{x}_j & \text{(DCOPY)} \\ \check{\boldsymbol{q}}_j \leftarrow L_j \check{\boldsymbol{q}}_j & \text{(DTRMV)} \\ \hat{\boldsymbol{q}}_j \leftarrow \hat{Y}_j \boldsymbol{x}_j + 0 \cdot \hat{\boldsymbol{q}}_j & \text{(DGEMV)} \\ q_{j,j} \leftarrow q_{j,j} - 1 & \end{cases},$$

where $\boldsymbol{x}_j \in \mathbb{R}^j$ is assigned on workspace memory.

When the above implementation is adapted, the highest order of the computational cost of the compact WY algorithm reduced to $4m^2n - m^3$. In the worst case, i.e., $m = n$, the computational cost of the new implementation of the compact WY algorithm is almost $3n^3$.

In addition, our implementation have not to be referred any zero elements of Y_j and T_j. Therefore, if Y_j and T_j are assigned on a CPU memory like Alg.1, the use of memory can be reduced to almost $n(m+1)$,

3.4. Comparison of the orthogonalization algorithms

The compact WY orthogonalization has a stable orthogonality arising from the Householder transformations, and its numerical computation is mainly performed by BLAS level-2 operations. As a result, this orthogonalization has a better stability and a sophisticated orthogonality, and it is more effective for parallel computing than MGS. Table 1 displays the differences in performance of the orthogonalization methods mentioned above. In this table, *Computation* denotes the order of the computational cost.

Table 1: Comparison of the orthogonalization methods [1] [11]

orthogonalization	*Computation*	*Synchronization*	*Orthogonality*
MGS	$2m^2n$	$O(m^2)$	$O(\varepsilon\kappa(V))$
Householder	$4m^2n$	$O(m^2)$	$O(\varepsilon)$
compact WY	$4m^2n+m^3$	$O(m)$	$O(\varepsilon)$
new compact WY	$4m^2n-m^3$	$O(m)$	$O(\varepsilon)$

Synchronization means the order of the number of synchronizations. *Orthogonality* indicates the norm $\|Q^\top Q-I\|$ and ε denotes the machine epsilon and $\kappa(V)$ is the condition number of V.

4. Inverse iteration algorithm with compact WY orthogonalization

Authors have proposed an alternative inverse iteration algorithm in [6]. This algorithm is based on the classical inverse iteration algorithm implemented in DSTEIN and we change the orthogonalization process of it from MGS to the compact WY orthogonalization that is described on Sec. 3.3.1. In addition, it is shown that this algorithm is faster than the classical inverse iteration one in parallel computing [6].

Now we present an even faster inverse iteration algorithm with the compact WY orthogonalization. This compact WY orthogonalization is implemented on the way of Sec. 3.3.2. The new algorithm is described in Alg.4. Let us name the new code DSTEIN-cWY.

Next, we explain an application of the new implementation of the compact WY orthogonalization to the inverse iteration. Differences between DSTEIN-cWY and DSTEIN is as follow: For the classical inverse iteration algorithm, we need not know the index j_c which denotes the j_c-th eigenvalue of the cluster in computing the eigenvector associated with it. However, we must know the index for the compact WY orthogonalization when we compute and update T_j, Y_j. To overcome the above difficulty, we introduce a variable j_c on line 9, and we can recognize it. This introduction of j_c enables us to execute the intended program.

In the classical inverse iteration algorithm, we need not know the first eigenvalue λ_{j_1} of the cluster. However, we must compute $\boldsymbol{y}_1$ and t_1 in the new inverse iteration algorithm. Therefore, at the starting point of the computation of the eigenvector associated with the second eigenvalue λ_{j_1+1}, we compute $T_1=[t_1]$, $Y_1=[\boldsymbol{y}_1]$ by using $\boldsymbol{v}_{j_1}$. At this time, because $\boldsymbol{v}_{j_1}$ is a normalized vector so that it equals to $(I-Y_1T_1Y_1^\top)\boldsymbol{e}_1$, we need not compute $\boldsymbol{v}_{j_1}$ it again.

5. Numerical experiments

We describe some numerical experiments performed by using DSTEIN and DSTEIN-cWY on parallel computers, and we compare the computation time. Here DSTEIN of LAPACK is based on the classical inverse iteration, and DSTEIN-cWY makes use of the new inverse iteration presented in the previous section.

Alg. 4 compact WY inverse iteration

for $j = 1$ to n **do**
  Generate $\boldsymbol{v}_j^{(0)}$ from random numbers.
  $k = 0$
  repeat
    $k \leftarrow k + 1$.
    Normalize $\boldsymbol{v}_j^{(k-1)}$.
    Solve $\left(T - \tilde{\lambda}_j I\right) \boldsymbol{v}_j^{(k)} = \boldsymbol{v}_j^{(k-1)}$.
    if $|\tilde{\lambda}_j - \tilde{\lambda}_{j-1}| \leq 10^{-3}\|T\|$, **then**
      $j_c \leftarrow j - j_1$.
      if $j_c = 1$ and $k = 1$, **then**
        Compute $Y_1 = [\boldsymbol{y}_1]$ and $T_1 = [t_1]$ by using $\boldsymbol{v}_{j_1}$.
      end if
      $\boldsymbol{u}_{j_c+1} = \left(I - Y_{j_c} T_{j_c}^\top Y_{j_c}^\top\right) \boldsymbol{v}_j^{(k)}$.
      Compute $\boldsymbol{y}_{j_c+1}$ and t_{j_c+1} by using $\boldsymbol{u}_{j_c+1}$.
      $Y_{j_c+1} = \begin{bmatrix} Y_{j_c} & \boldsymbol{y}_{j_c+1} \end{bmatrix}$, $T_{j_c+1} = \begin{bmatrix} T_{j_c} & -t_{j_c+1} T_{j_c} Y_{j_c}^\top \boldsymbol{y}_{j_c+1} \\ \boldsymbol{0}_{j_c}^\top & t_{j_c+1} \end{bmatrix}$.
      $\boldsymbol{v}_j^{(k)} \leftarrow \left(I - Y_{j_c+1} T_{j_c+1} Y_{j_c+1}^\top\right) \boldsymbol{e}_{j_c+1}$.
    else
      $j_1 \leftarrow j$.
    end if
  until Some condition is met.
  Normalize $\boldsymbol{v}_j^{(k)}$ to $\boldsymbol{v}_j$.
end for

5.1. Contents of the numerical experiments

We report computations of all the eigenvectors associated with eigenvalues of some matrices by using DSTEIN and DSTEIN-cWY on parallel computers, and we compare the elapsed time. In these experiments, we compute the approximate eigenvalues by using LAPACK's program DSTEBZ, which is capable of computing eigenvalues using the bisection method. We record the elapsed time for DSTEIN and DSTEIN-cWY using SYSTEM_CLOCK, which is the internal function of Fortran.

In the experiments, we use two computers equipped with multicore CPUs, and we implement those algorithms by using GotoBLAS2 [5], which is implemented to parallelize BLAS operations by assigning them to each CPU core. Table 2 shows the specifications of two computers. As experimental matrices, we use symmetric tri-diagonal matrices of three types. Type 1 is a tri-diagonal random matrix, of which elements are set to the random number of $[0,1)$. It is shown that the eigenvalues of a tri-diagonal random matrix are divided into a few clusters in the sense of Peters-Wilkinson method[8]. and most of eigenvalues are included in the biggest one of the clusters if the dimension n of a random matrix becomes larger. The tri-diagonal matrix

Table 2: The specification of Computer 1 and 2

	Computer 1	Computer 2
CPU	AMD Opteron 2.0GHz 32cores(8cores×4)	Intel Xeon 2.93GHz 8cores(4cores×2)
RAM	256GB	32GB
Compiler	Gfortran-4.4.5	Gfortran-4.4.5
LAPACK	LAPACK-3.3.0	LAPACK-3.3.0
BLAS	GotoBLAS2-1.13	GotoBLAS2-1.13

of Type 2 is defined as follows:

$$T = \begin{bmatrix} 1 & 1 & & & & \\ 1 & 1 & 1 & & & \\ & 1 & \ddots & \ddots & & \\ & & \ddots & \ddots & 1 & \\ & & & 1 & 1 & \end{bmatrix}. \tag{5}$$

All the eigenvalues of Type 2 matrix with large dimensions are included in the same cluster in the sense of Peters-Wilkinson method. Type 3 is the glued-Wilkinson matrices $W_g^\dagger$. $W_g^\dagger$ consists of the block matrix $W_{21}^\dagger \in \mathbb{R}^{21\times 21}$ and the scalar parameter $\delta \in \mathbb{R}$ and is defined as follow:

$$W_g^\dagger = \left[\begin{array}{c|c|c|c} W_{21}^\dagger & \delta & & \\ \hline \delta & W_{21}^\dagger & \delta & \\ \hline & \delta & \ddots \; \ddots & \\ & & \ddots \; \ddots & \delta \\ \hline & & \delta & W_{21}^\dagger \end{array}\right], \tag{6}$$

where $W_{21}^\dagger$ is defined by

$$W_{21}^\dagger = \begin{bmatrix} 10 & 1 & & & & & \\ 1 & 9 & 1 & & & & \\ & 1 & \ddots & \ddots & & & \\ & & \ddots & 0 & \ddots & & \\ & & & \ddots & \ddots & 1 & \\ & & & & 1 & 10 & \end{bmatrix}, \tag{7}$$

and δ satisfies $0 < \delta < 1$ and is also the semi-diagonal element of $W_g^\dagger$. Since $W_g^\dagger$ is real symmetric tri-diagonal and its semi-diagonal elements are nonzero, all the eigenvalues of $W_g^\dagger$ are real and they are divided into 21 clusters of close eigenvalues. When δ is

Table 3: Numerical results of DSTEIN and DSTEIN-cWY on Computer 1 (Type 1).

n	1050	2100	3150	4200	5250	6300	7350	8400	9450	10500
t [sec.]	0.39	1.76	5.30	17.4	53.6	157	996	2436	4004	13231
t_{cwy} [sec.]	0.41	1.60	3.77	7.85	13.7	25.1	115	307	449	1291
t/t_{cwy}	0.94	1.10	1.41	2.22	3.90	6.22	8.64	7.93	8.93	10.25

Table 4: Numerical results of DSTEIN and DSTEIN-cWY on Computer 2 (Type 1).

n	1050	2100	3150	4200	5250	6300	7350	8400	9450	10500
t [sec.]	0.16	0.75	2.13	6.41	19.2	58.3	372	889	1416	4357
t_{cwy} [sec.]	0.18	0.73	1.70	3.42	7.66	24.7	179	430	703	1933
t/t_{cwy}	0.91	1.02	1.25	1.87	2.51	2.36	2.08	2.06	2.01	2.25

Table 5: Numerical results of DSTEIN and DSTEIN-cWY on Computer 1 (Type 2).

n	1050	2100	3150	4200	5250	6300	7350	8400	9450	10500
t [sec.]	1.73	154	448	989	1897	3281	5192	7749	10986	14867
t_{cwy} [sec.]	0.45	7.04	28.1	94.6	167	311	476	795	1029	1389
t/t_{cwy}	3.85	21.93	15.94	10.45	11.34	10.56	10.92	9.74	10.68	10.70

Table 6: Numerical results of DSTEIN and DSTEIN-cWY on Computer 2 (Type 2).

n	1050	2100	3150	4200	5250	6300	7350	8400	9450	10500
t [sec.]	0.52	57.4	171	375	688	1143	1774	2570	3586	4884
t_{cwy} [sec.]	0.20	12.2	55.3	136	266	462	723	1067	1519	2070
t/t_{cwy}	2.67	4.69	3.10	2.75	2.58	2.48	2.45	2.41	2.36	2.36

Table 7: Numerical results of DSTEIN and DSTEIN-cWY on Computer 1 (Type 3).

n	1050	2100	3150	4200	5250	6300	7350	8400	9450	10500
t [sec.]	2.26	11.5	31.8	72.9	138	230	359	526	738	986
t_{cwy} [sec.]	0.62	2.49	5.82	10.9	18.1	28.4	45.9	74.5	103	141
t/t_{cwy}	3.66	4.62	5.47	6.71	7.66	8.10	7.82	7.06	7.18	6.99

Table 8: Numerical results of DSTEIN and DSTEIN-cWY on Computer 2 (Type 3).

n	1050	2100	3150	4200	5250	6300	7350	8400	9450	10500
t [sec.]	0.68	3.58	10.4	24.5	50.1	86.8	137	203	289	393
t_{cwy} [sec.]	0.27	1.10	2.72	6.59	16.9	35.7	63.4	103	149	209
t/t_{cwy}	2.54	3.27	3.83	3.72	2.97	2.43	2.16	1.97	1.94	1.88

small, the distance between the minimum and maximum eigenvalues in any cluster is small. In our experiments, we set $\delta = 10^{-4}$. Computing eigenvalues and eigenvectors of the glued-Wilkinson matrix is one of the benchmark problems of eigenvalue decomposition. For example, the glued-Wilkinson matrix was used to evaluate the performance of matrix eigenvalue algorithms [2] [4].

5.2. Results of the experiments

Table 3-8 show the results of the experiments on Computer 1 and 2 that are mentioned in the previous section, In tables, n is the dimension of the experimental matrices, t and t_{cwy} are computation time by DSTEIN and DSTEIN-cWY, respectively.

In addition, Fig. 2-4 illustrate the results in Tables 3 and 4, 5 and 6, 7 and 8 through graphs, respectively. In Fig. 2-4, the dotted line corresponds to t and the straight line to t_{cwy}.

It is noted that DSTEIN-cWY is faster than DSTEIN for any cases of the all types matrices, without the cases of Type 1 matrix for $n = 1050$. We see that the change from MGS to the compact WY orthogonalization on the DSTEIN code in parallel computing results in a significant reduction of computation time. We introduce a barometer t/t_{cwy} of the reduction effect by using the program DSTEIN-cWY which depends on n, the dimension of the experimental matrix. On Computer 1, the maximum value of $\alpha = t/t_{\mathrm{cwy}}$ is $\alpha = 10.25$ for $n = 10,500$ of Type 1, $\alpha = 10.92$ for $n = 7,350$ of Type 2, and $\alpha = 8.10$ for $n = 6,300$ of Type 3. On Computer 2, $\alpha = 2.51$ for $n = 5,250$ of Type 1, $\alpha = 4.69$ for $n = 2,100$ of Type 2, and $\alpha = 3.83$ for $n = 3,150$ of Type 3. Considering these facts, even if the dimension of the experimental matrices is larger than that in these examples, we cannot expect that the computation time can be further shortened by using DSTEIN-cWY.

5.3. Discussion on numerical experiments

It is shown that DSTEIN-cWY is faster than DSTEIN for any dimension n of the experimental matrix both on Computers 1 and 2. As mentioned earlier, according to the theoretical background in Section 3.3, this result shows that the compact WY orthogonalization is an effective algorithm for parallel computing.

The cause of this is related to the time required for floating-point arithmetic and for synchronization in parallel computing. The floating-point computation time increases with increasing the dimension n of matrices. In comparison, the synchronization cost does not change significantly even if n becomes larger. Therefore, in parallel computing, DSTEIN, which contains MGS (for which the number of synchronizations is large), creates a huge bottleneck for the synchronization cost when n is small. This bottleneck gradually becomes less when n is larger. However, DSTEIN-cWY has a smaller bottleneck for the synchronization cost because the compact WY orthogonalization requires less synchronization, and the floating-point computation time becomes greater than that of DSTEIN. This reduction effect can be seen in Table 3-8.

6. Conclusions

In this study, we present a new inverse iteration algorithm for computing all the eigenvectors of a real symmetric tri-diagonal matrix. The new algorithm is equipped with the new implementation of the compact WY orthogonalization algorithm, established in this paper, in the orthogonalization process.

Now we use a new implementation of the compact WY orthogonalization. Introducing this implementation, the computational cost of the compact WY orthogonalization can be reduced.

We have given numerical experiments for computing eigenvectors of certain real symmetric tri-diagonal matrices that have many clusters with several thousand dimensions by using two types of inverse iteration algorithms on parallel computers. The results show that the compact WY inverse iteration is more efficient than the classical

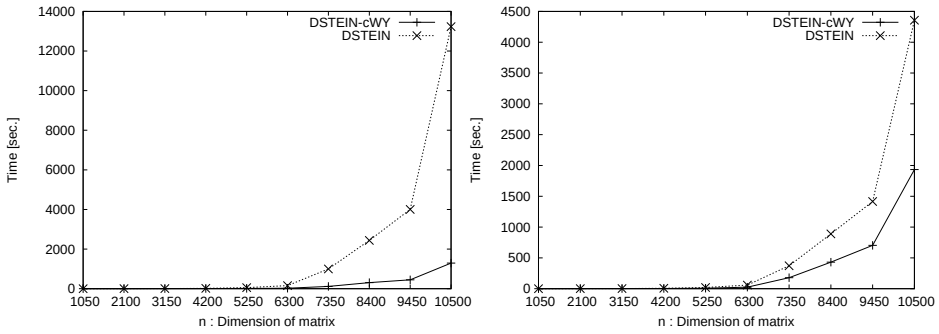

Figure 2: Dimension n of Type 1 matrix and the computation time by DSTEIN and DSTEIN-cWY. the left graph corresponds to Computer 1 and the right Computer 2.

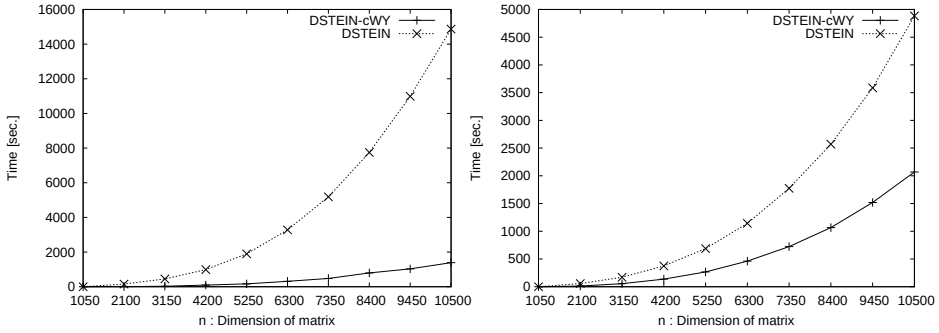

Figure 3: Dimension n of Type 2 matrix and the computation time by DSTEIN and DSTEIN-cWY. the left graph corresponds to Computer 1 and the right Computer 2.

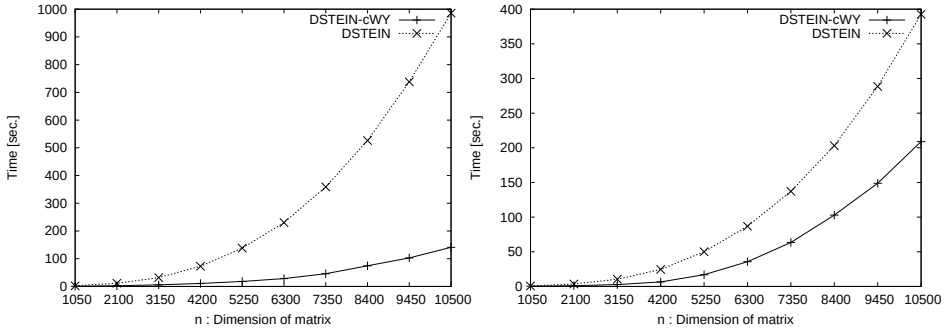

Figure 4: Dimension n of Type 3 matrix and the computation time by DSTEIN and DSTEIN-cWY. the left graph corresponds to Computer 1 and the right Computer 2.

one owing to the reduction in computation time because of the parallelization efficiency. As the number of cores of the CPU increases, the parallelization efficiency increases.

It may be expected to apply the new inverse iteration algorithms to other types of matrix eigenvector problem, such as eigenvectors of a real symmetric band matrix, or

singular vectors of a bidiagonal matrix.

References

[1] J. W. Demmel, L. Grigori, M. Hoemmen and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, LAPACK Working Notes, No.204, 2008.

[2] J. W. Demmel, O. A. Marques, B. N. Parlett, and C. Vömel, *Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers*, SIAM J. Sci. Comput., Vol. 30, No. 3, pp. 1508-1526, 2008.

[3] I. S. Dhillon, *A new O(n^2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*, Ph.D. thesis, Computer Science Division, University of California, Berkeley, California, UC Berkeley Technical Report UCB//CSD-97-971, 1997.

[4] I. S. Dhillon, B. N. Parlett, and C. Vömel, *Glued matrices and the MRRR algorithm*, SIAM J. Sci. Comput., Vol. 27, No. 2, pp. 496-510, 2005.

[5] GotoBLAS2, `http://www.tacc.utexas.edu/tacc-projects/gotoblas2/`.

[6] H. Ishigami, K. Kimura and Y. Nakamura, *Implementation and Performance Evaluation of New Inverse Iteration Algorithm with Householder Transformation in Terms of the Compact WY Representation*, Proc. of the 2011 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2011), Vol. II, pp. 775-780, 2011.

[7] LAPACK, `http://www.netlib.org/lapack/`.

[8] G. Peters and J. Wilkinson, *The calculation of specified eigenvectors by inverse iteration*, contribution II/18, in Linear Algebra, Handbook for Automatic Computation, Vol. II, Springer-Verlag, Berlin, pp. 418-439, 1971.

[9] R. Schreiber and C. van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., Vol. 10, No. 1, pp. 53-57, 1989.

[10] H. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., Vol. 9, No. 1, pp. 152-163, 1988.

[11] Y. Yamamoto and Y. Hirota, *A parallel algorithm for incremental orthogonalization based on the compact WY representation*, JSIAM Letters, Vol. 3, pp. 89-92, 2011.